EECS 336: Lecture 2: Introduction to Algorithms

Philosophy, Tractibility, Big-Oh

Reading: Chapters 2 & 3.

Announcements:

- Lecture notes on Canvas (before class.)
- Practice on "solved problems" in text.
- Prerequisites:
  - EECS 212: Discrete math.
  - EECS 214: Data Structures.
- Homework:
  - work with partner
  - must communicate solution well.
  - automatically drop 3 lowest hw grades & 3 lowest peer reviews.
  - peer review
    - can you tell if algorithm and proof are correct?
    - communicate algorithms.
  - solutions Wednesday, peer reviews Friday, grades Monday.
- Peer review logisitics
  - reviews assigned Thursday morning, due Friday evening.
  - 3 peer review per problem.
  - 1 peer review is graded (random)
  - detail rubric provided.
- Exam dates on Canvas.

Last Time:

- motivation
- fibonacci numbers

Today:

- philosophy
- computational tractability
- runtime analysis & big-oh

Algorithm Design and Analysis

gives rigorous mathematical framework for thinking about and solving problems in CS and other fields.

Goals

- quickly compute solutions to problems.
- identify general algorithm design and analysis approaches.
- understand what makes problems tractable or intractable.

Three Steps

1. problem modeling: abstract problem to essential details.
2. algorithm design
3. algorithm analysis
    - efficiency
    - correctness, and
    - (sometimes) "quality."

Computational Tractability

"is a problem solvable by a computer?"

Def: problem is *tractable* if worst-case run-time to compute so lution is polynominal in size of input.

Def: T(n) = worst case runtime of instances of size n.

- size n measured in bits, or
- number of "components."

Example: Fibonacci Numbers

fib(k) has n = log k bits.

- recursive : $T(n) \approx 2^{2^n}$
- dynamic program / iterative alg: $T(n) \approx 2^n$
- repeated squaring: $T(n) \approx n$.

Question: why worst case?

- every instance?
- typical instances?
- random instances?

Question: Benefits?

- usually doable.
- often tight for typical or random instances.
- analyses "compose"

Question: why polynomial?

Answer: polynomial means algorithm scales well, i.e., $T(cn) \leq dT(n)$.

Example:

$$\begin{aligned} T(n) =& n^k \\ T(cn) =& (cn)^k = \underbrace{c^k}_{\text{d}} n^k = dn^k \end{aligned}$$

Tractable vs. Brute-force

- brute-force: "try all solutions, output best one"
- often runtime of brute-force $\geq$ exponential time.
- tractable algorithms require exploiting structure of problem.

Main goals for algorithm design

1. show problem is tractable: exists algorithm with polynomial runtime.
2. show problem is intractable for all algorithms, runtime is super-polynomial.

Question: Which is easier?

Answer: showing tractable.

Runtime Analysis

"bound $T(n)$ for algorithm"

Big-Oh Notation

Def: $T(n)$ is $O(f(n))$ if

$\exists n_0, c > 0$ such that $\forall n > n_0$, $T(n) < cf(n)$.

Question: why?

Answer:

- exact analysis is too detailed.
- constants vary from machine to machine.

Example:

$$\begin{aligned} T(n) =& an^2 + bn + d \\ =& O(n)?\ O(n^2)?\ O(n^3)? \\ T(n) \leq& an^2 + bn^2 + dn^2 \\ =& \underbrace{(a+b+d)}_{c} n^2 \\ \leq& cn^2 \end{aligned}$$

Fact 1: $f = O(g)$ & $g = O(h) \implies f = O(h)$

Fact 2: $f = O(h)$ & $g = O(h) \implies f + g = O(h)$

Fact 3: $g = O(f) \implies g + f = O(f)$

Proof: (of Fact 2)

$f = O(h) \implies \exists c, n_0$ such that $\forall n > n_0, f(n) < ch(n)$

$g = O(h) \implies \exists c', n_0'$ such that $\forall n > n_0', g(n) < c'h(n)$

$\implies \forall n \geq \max(n_0, n_0'), f(n) + g(n) \leq (c' + c)h(n)$

$\implies f + g = O(n)$

QED

Note:

- be succinct: do not write $O(n^2 + 2)$, $O(5n)$, etc.
- be tight: if T(n) is n^2 do not say T(n) is $O(n^3)$.

Logarithms and Big-Oh

Def: $\log_b n = l \leftrightarrow b^l = n$

- $\log_{10} n =$ number of digits to represent n.
- $\log_2 n =$ number of bits to represent n.

Fact 4: $\forall b, c\ \log_b = O(\log_c n)$

Fact 5: $\forall b, x\ \log_b n = O(n^x)$.

Proof: (of Fact 4)

$$\begin{aligned} \log_c n = l &\implies n = c^l \\ \log_b n &= \log_b(c^l) \\ &= l \log_b(c) \\ &= \log_c n \underbrace{\log_b c}_{d} \\ &= O(\log_c n) \end{aligned}$$

QED

Common Runtimes

- $O(\log n)$ - logarithmic
- $O(n)$ - linear
- $O(nlogn)$
- $O(n^2)$ - quadratic
- $O(n^3)$ - cubic
- $O(n^k)$ - polynomial
- $O(2^n)$ - exponential
- $O(n!)$ - Uh-Oh

Lower Bounds

Def: $T(n)$ is $\Omega(f(n))$ if

$\exists n_0, c > 0$ such that $\forall n > n_0, T(n) > cf(n)$.

Exact Bounds

Def: $T(n)$ is $\Theta(f(n))$ if

$T(n)$ is $O(f(n))$ and $\Omega(f(n))$.

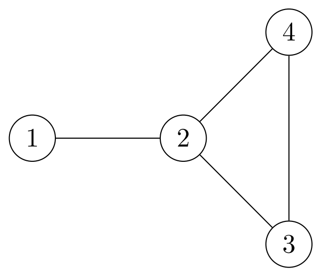
Graphs

"encode pair-wise relationships"

Examples: computer networks, social networks, travel networks, dependencies.

$$G = (\underbrace{V}_{\text{vertices}}, \overbrace{E}^{\text{edges}})$$

Example:



- $V = \{1, 2, 3, 4\}$
- $E = \{(1,2), (2,3), (3,4), (2,4)\}$

Concepts

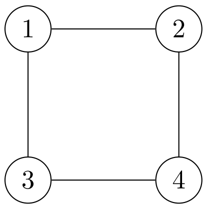
- degree
- neighbors
- paths, path length
- distance
- connectivity, connected components
- directed graphs

Graph Traversals

"visit all the vertices in a connected component of graph"

- Breadth First Search (BFS).

Example:



BFS from 1: 1, 2, 3, 4 or 1, 3, 2, 4.

- Depth First Search (DFS).

Example: DFS from 1: 1, 2, 4, 3 or 1, 3, 4, 2.